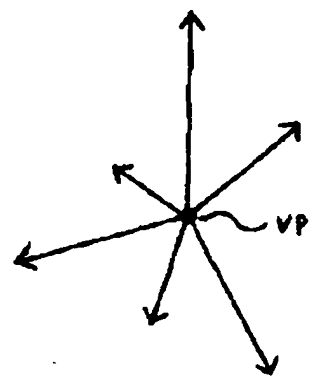


FIGURE 1.

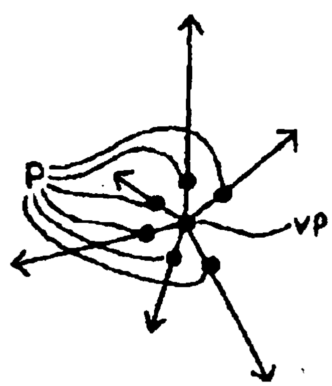


FIGURE 2.

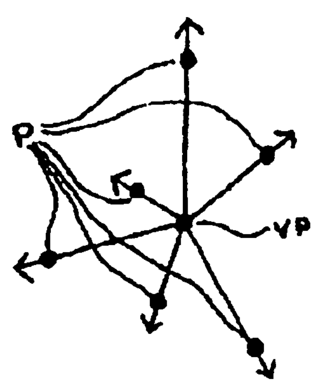


FIGURE 3.

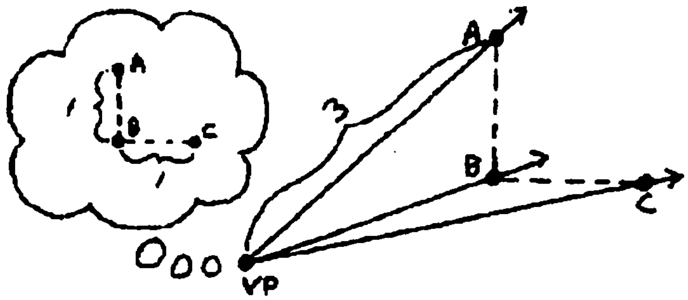


FIGURE 4A.

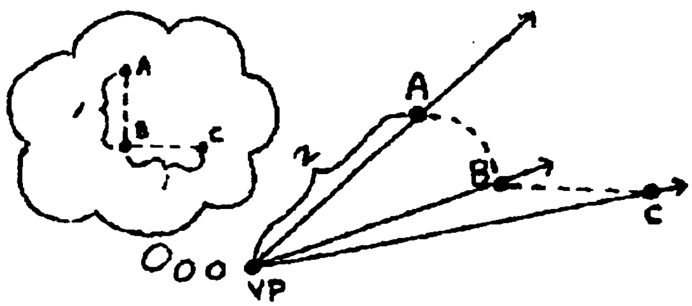


FIGURE 4B.

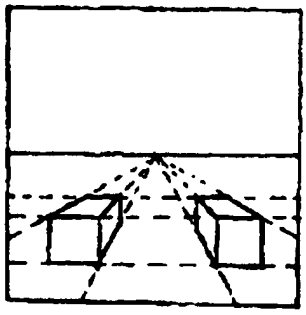


FIGURE 5.

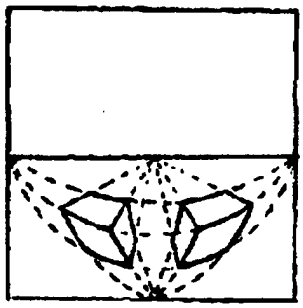


FIGURE 6.

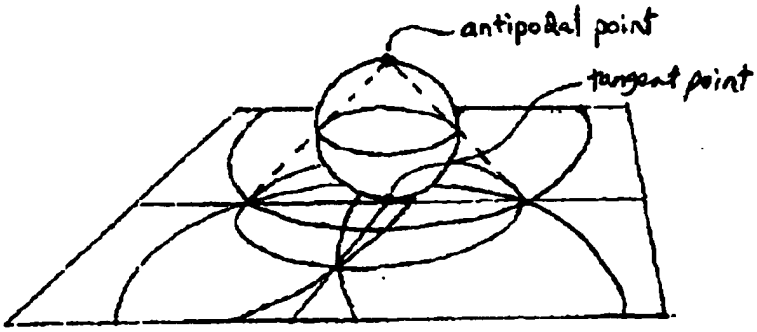
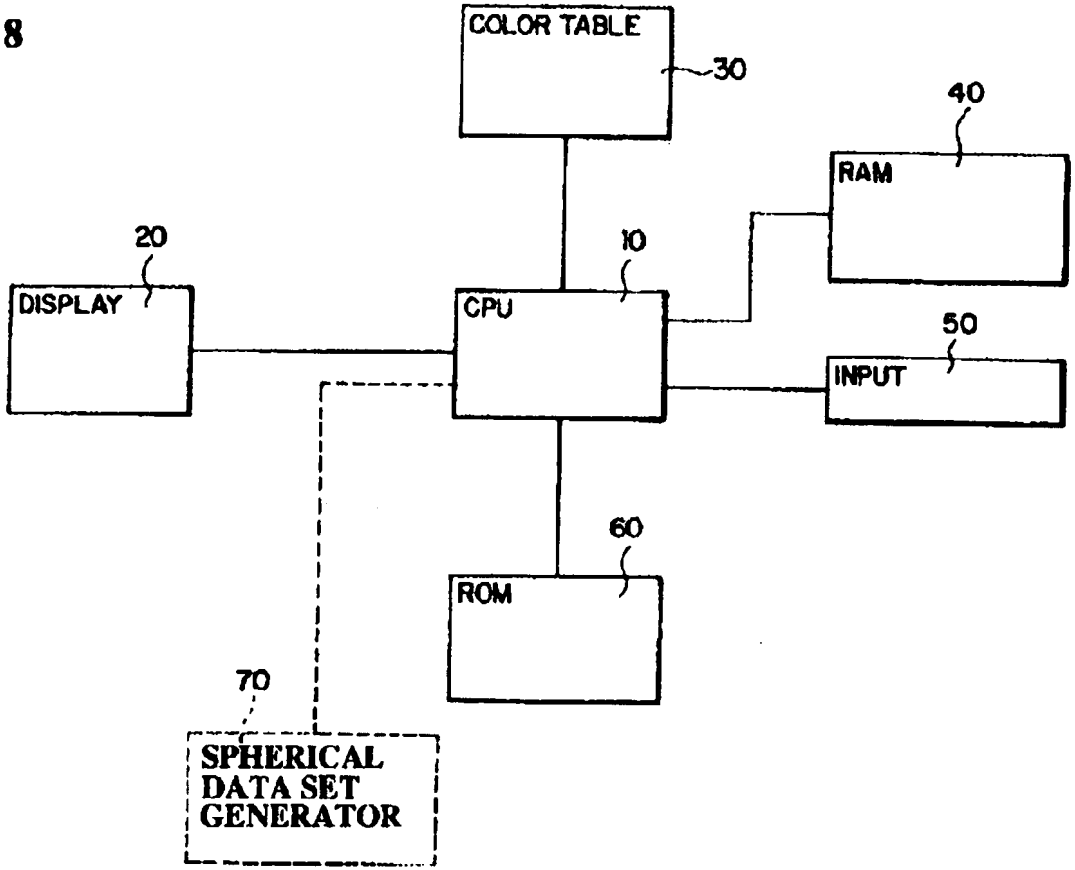


FIGURE 7.

Fig. 8



```
/* Includes required */
#include <GL/gl.h>
#include <GL/glut.h>
#include <stdio.h>
#include <ppm.h>
#include <math.h>

/**
 * something because of windows
 */
void __eprintf() {
}

/**
 * our data structure of choice
 */
typedef struct obj {
    /* other parameters */
    float matrix[16];

    /* view angle */
    float viewangle;

    /* aspect ratio */
    float aspect;

    /* z of the camera */
    float tz;

    /* ry of the camera */
    float ry;
} Obj;

/* hold the display lists for textures */
typedef struct texture {
    int tex1;
    int tex2;
} Texture;

/**
 * our global variables
 */
/* camera settings */
Obj scene;

/* texture stuff */
Texture def;
Texture* current_texture = &def;

/* track the next display list number */
int nextDLnum = 2;

/* stuff for lighting */
float lightPos[4] = {2.0, 4.0, 2.0, 0};
float lightDir[4] = {0, 0, 1.0, 1.0};
float lightAmb[4] = {0.4, 0.4, 0.4, 1.0};
float lightDiff[4] = {0.8, 0.8, 0.8, 1.0};
float lightSpec[4] = {0.8, 0.8, 0.8, 1.0};
int lights = 0;
int outsideView = 0;
int parent;


#define HEMISPHERE 1
void createHemisphere(int listNum, int numPts, int geom);
```

Fig. 9A

```
/**
 * Read in the ppm files and create display lists for a texture
 * returns the dimension of the image
 */
pixel **map1, **map2;
GLubyte *tex1, *tex2, **tmpPP, *tmpP;
void readTexture(Texture* t, char* file1, char* file2) {
    FILE *fp1, *fp2;
    int cols, rows, i, j, index;
    pixval maxval;

    /* open the files */
    fp1 = fopen(file1, "r");
    fp2 = fopen(file2, "r");
    if (!fp1) {
        fprintf(stderr, "Couldn't open %s\n", file1);
    }
    if (!fp2) {
        fprintf(stderr, "Couldn't open %s\n", file2);
    }

    /* read the ppm files */
    map1 = ppm_readppm(fp1, &cols, &rows, &maxval);
    fprintf(stderr, "%s: rows = %d \t cols = %d\n", file1, rows, cols, maxval);
    map2 = ppm_readppm(fp2, &cols, &rows, &maxval);
    fprintf(stderr, "%s: rows = %d \t cols = %d\n", file2, rows, cols, maxval);

    /* convert them */
    tex1 = malloc(sizeof(GLubyte) * rows * cols * 3);
    tex2 = malloc(sizeof(GLubyte) * rows * cols * 3);
    index = 0;
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            /* R */
            tex1[index] = PPM_GETR(map1[i][j]);
            tex2[index] = PPM_GETR(map2[i][j]);
            index ++;

            /* G */
            tex1[index] = PPM_GETG(map1[i][j]);
            tex2[index] = PPM_GETG(map2[i][j]);
            index ++;

            /* B */
            tex1[index] = PPM_GETB(map1[i][j]);
            tex2[index] = PPM_GETB(map2[i][j]);
            index ++;
        }
    }

    /* create the textures */
    /* new display list*/
    glNewList(nextDLnum, GL_COMPILE);
    t->tex1 = nextDLnum;
    nextDLnum++;
    glTexImage2D(GL_TEXTURE_2D, 0, 3, cols, rows, 0, GL_RGB, GL_UNSIGNED_BYTE,
                 tex1);
    glEndList();

    /* new display list*/
    glNewList(nextDLnum, GL_COMPILE);
    t->tex2 = nextDLnum;
    nextDLnum++;
    glTexImage2D(GL_TEXTURE_2D, 0, 3, cols, rows, 0, GL_RGB, GL_UNSIGNED_BYTE,
```

Fig. 9B

```
                tex2);
    glEndList();
}

/**
 * this will initialize the display lists for the objects
 */
void initialize_objects(int argc, char**argv) {
    float tmp[4];

    /* read in the texture */
    readTexture(&def, argv[1], argv[2]);

    /* create hemisphere */
    createHemisphere(1, 50, GL_TRIANGLE_STRIP);

    /* scene */
    scene.viewangle = 130;
    scene.tz = 0;
    scene.ry = 0;
}
```

Fig. 9C

```
/*
 *  Clear the screen. draw the objects
 */
void display()
{
    float tmp[4];
    float height;

    /* clear the screen */
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    /* adjust for scene orientation */
    glMatrixMode(GL_PROJECTION);
    if (outsideView) {
        glLoadIdentity();
        gluPerspective(45, scene.aspect, 0.1, 10.0);
        glTranslatef(0, C, -3);
        glRotatef(45, 1, 0, 0);
        glRotatef(45, 0, 1, 0);
        glDisable(GL_TEXTURE_2D);
        glColor3f(.8, .8, .8);
    } else {
        glLoadIdentity();
        gluPerspective(scene.viewangle, scene.aspect, 0.1, 10.0);
        glTranslatef(0, 0, scene.tz);
        glRotatef(scene.ry, 0, 1, 0);
    }

    /* draw our models */
    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();

    if (outsideView) {
        /* transform to where the camera would be */
        glPushMatrix();

        /* draw a cube for the camera */
        glLoadIdentity();
        glRotatef(180, 1, 0, 0);
        glTranslatef(0, 0, scene.tz);
        tmp[0] = tmp[1] = tmp[2] = .8;
        tmp[3] = 1;
```

```
        glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, tmp);
        glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 0.0);
        glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, tmp);
        glutSolidCube(.1);

        /* draw a cone for the view frustrum */
        glLoadIdentity();
        height = 1 - scene.tz;
        glRotatef(45, 0, 0, 1);
        glTranslatef(0, 0, -1);
        tmp[0] = tmp[1] = 1;
        tmp[2] = 0;
        tmp[3] = .3;
        glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, tmp);
        glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 0.0);
        glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, tmp);
        glutSolidCone(tan(scene.viewangle * 3.14 / 360.0) * height, height, 20, 1);
        glPopMatrix();
        glEnable(GL_TEXTURE_2D);
    }

    /* now draw the semisphere */
    if (lights) {
        tmp[0] = tmp[1] = tmp[2] = .8;
        tmp[3] = .8;
        glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, tmp);
        glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 10.0);
        glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, tmp);
    }

    glCallList(current_texture->tex1);
    glCallList(HEMISPHERE);

    if (lights) {
        tmp[0] = tmp[1] = tmp[2] = .5;
        tmp[3] = .5;
        glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, tmp);
        glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 10.0);
        glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, tmp);
    }

    glRotatef(180.0, 0.0, 0.0, 1.0);
    glCallList(current_texture->tex2);
    glCallList(HEMISPHERE);
    glPopMatrix();

    fprintf(stderr, "%s\n", gluErrorString(glGetError()));
    glutSwapBuffers();
}

/*
 *  Handle Menus
 */
#define M_QUIT 1
void Select(int value)
{
    switch (value) {
    case M_QUIT:
        exit(0);
        break;
    }
    glutPostRedisplay();
}
void create_menu() {
    fprintf(stderr, "Press ? for help\n");
```

Fig. 9D

Fig. 9E

```
    glutCreateMenu(Select);
    glutAddMenuEntry("Quit", M_QUIT);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
}


/* Initializes hading model */
void myInit(void)
{
    glEnable(GL_DEPTH_TEST);
    glShadeModel(GL_SMOOTH);

    /* texture stuff */
    glPixelStorei(GL_UNPACK_ALIGNMENT, sizeof(GLubyte));
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
    glEnable(GL_TEXTURE_2D);

}

/*
 *  Called when the window is first opened and whenever
 *  the window is reconfigured (moved or resized).
 */
void myReshape(int w, int h)
{
    glViewport (0, 0, w, h);            /*  define the viewport */
    scene.aspect = 1.0*(GLfloat)w/(GLfloat)h;
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(scene.viewangle, scene.aspect, 0.1, 10.0);
    glMultMatrixf(scene.matrix);
    glMatrixMode (GL_MODILVIEW);        /* back to modelview matrix  */
}

/*
 * Keyboard handler
 */
void
Key(unsigned char key, int x, int y)
{
    float matrix[16];
    glMatrixMode(GL_MODELVIEW);
    glGetFloatv(GL_MODELVIEW_MATRIX, matrix);
    glLoadIdentity();
    fprintf(stderr, "%d - %c  ", key, key);
    switch (key) {
    case 'o':
        if (!outsideView  {
            fprintf(stderr, "outside on  ");
            outsideView = 1;

            /* turn on blending */
            glEnable(GL_BLEND);
            glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

            /* We want to see color */
            glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

            /* turn on our spotlight */
            glEnable(GL_LIGHT1);
            glLightfv(GL_LIGHT1, GL_AMBIENT, lightAmb);
```

```
            glLightfv(GL_LIGHT1, GL_DIFFUSE, lightDiff);
            glLightfv(GL_LIGHT1, GL_SPECULAR, lightSpec);
            glLightfv(GL_LIGHT1, GL_SPOT_DIRECTION, lightDir);
        } else {
            fprintf(stderr, "outside off  ");
            outsideView = 0;
            glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
            glDisable(GL_BLEND);
        }
        break;
    case 'F':
        fprintf(stderr, "flat  ");
        glShadeModel(GL_FLAT);
        break;
    case 'f':
        fprintf(stderr, "smooth  ");
        glShadeModel(GL_SMOOTH);
        break;
    case 'y':
        printf("ry = %f\n", scene.ry);
        scene.ry -= 5;
        break;
    case 'Y':
        scene.ry += 5;
        break;
    case 'z':
        scene.tz -= .02;
        fprintf(stderr, " tz = %f  ", scene.tz);
        break;
    case 'Z':
        scene.tz += .02;
        fprintf(stderr, " tz = %f  ", scene.tz);
        break;
    case 'a':
        scene.viewangle -= 1;
        fprintf(stderr, " angle: %f  ", scene.viewangle);
        break;
    case 'A':
        scene.viewangle += 1;
        fprintf(stderr, "angle: %f  ", scene.viewangle);
        break;
    case 55:
        glRotatef(-5, 0.0, 0.0, 1.0);
        break;
    case 57:
        glRotatef(5, 0.0, 0.0, 1.0);
        break;
    case 52:
        glRotatef(-5, 0.0, 1.0, 0.0);
        break;
    case 54:
        glRotatef(5, 0.0, 1.0, 0.0);
        break;
    case 56:
        glRotatef(5, 1.0, 0.0, 0.0);
        break;
    case 50:
        glRotatef(-5, 1.0, 0.0, 0.0);
        break;
    case 'q':
        if (lights) {
            glDisable(GL_LIGHT0);
            glDisable(GL_LIGHTING);
            lights = 0;
            fprintf(stderr, "no lights  ");
```

Fig. 9F

```
        } else {
            glEnable(GL_LIGHTING);
            glEnable(GL_LIGHT0);
            glLightfv(GL_LIGHT0, GL_POSITION, lightPos);
            glLightfv(GL_LIGHT0, GL_AMBIENT, lightAmb);
            glLightfv(GL_LIGHT0, GL_DIFFUSE, lightDiff);
            glLightfv(GL_LIGHT0, GL_SPECULAR, lightSpec);
            lights = 1;
            fprintf(stderr, "lights  ");
        }
        break;
    case 't':
        fprintf(stderr, "texture off  ");
        glDisable(GL_TEXTURE_2D);
        break;
    case 'T':
        fprintf(stderr, "texture on  ");
        glEnable(GL_TEXTURE_2D);
        break;
    case '?':
        fprintf(stderr, "hjkl - rotate current object\n");
        fprintf(stderr, "s/S - shrink / grow the object or zoom the scene\n");
        fprintf(stderr, "a/A viewangle\n");
        fprintf(stderr, "z/Z camera position\n");
        fprintf(stderr, "f/F flat smooth\n");
        fprintf(stderr, "Escape quits \n");
        break;
    case 27:            /* Esc will quit */
        exit(1);
        break;
    default:
        fprintf(stderr, "Unbound key - %d  ", key);
        break;
    }
    fprintf(stderr, "\n");
    glMultMatrixf(matrix);
    glutPostRedisplay();
}

/*
 *  Main Loop
 *  Open window with initial window size, title bar,
 *  RGBA display mode, and handle input events.
 */
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGBA);
    parent = glutCreateWindow (argv[0]);
    myInit();
    glutKeyboardFunc(Key);
    glutReshapeFunc (myReshape);
    glutDisplayFunc(display);
    create_menu();
    initialize_objects(argc, argv);
    glutMainLoop();
}
```

Fig. 9G

Fig. 10A

```c
#ifdef WINDOWS
#include <windows.h>
#endif
#include <GL/gl.h>
#include <GL/glut.h>

#include "warp.h"
#include <stdio.h>
/**
 * Triangulate a hemisphere and texture coordinates.
 * listNum - display list number
 * numPts -  number of points to a side
 * return the display list
 */
void createHemisphere(int listNum, int numPts, int geom) {
    double incr = 1.0 / numPts;
    double u, v, x, y, z;
    float tx, tz;
    int i, j;

    /* start the display list */
    glNewList(listNum, GL_COMPILE_AND_EXECUTE);

    /* create the coordinates */
    /* use the square to circle map */
    /* across then down */
    v = 0;
    for (j = 0; j < numPts ; j++) {
        /* start the tri strip */
        glBegin(geom);
        u = 0;
        for (i = 0; i <= numPts; i++) {
            /* do the top point */
            /* get the XYZ coords */
            map(u, v, &x, &y, &z);

            /* create the texture coord */
            tx = x / 2 + .5;
            tz = z / 2 + .5;
            if (tx > 1.0 || tz > 1.0 || tx < 0.0 || tz < 0.0) {
                printf("not in range %f %f\n", tx, tz);
            }
            glTexCoord2f(tx, tz);

            /* normal */
            glNormal3f(x, y, z);

            /* create the coord */
            glVertex3f(x, y, z);

            /* get the XYZ coords */
            map(u, v + incr, &x, &y, &z);

            /* create the texture coord */
            tx = x / 2 + .5;
            tz = z / 2 + .5;
            if (tx > 1.0 || tz > 1.0 || tx < 0.0 || tz < 0.0) {
                printf("not in range %f %f\n", tx, tz);
            }
            glTexCoord2f(tx, tz);

            /* normal */
            glNormal3f(x, y, z);

            /* create the coord */
```

```
            glVertex3f(x, y, z);

            /* adjust u */
            u += incr;
        }
        /* done with the list */
        glEnd();

        /* adjust v */
        v += incr;
    }

    /* all done with the list */
    glEndList();
}
```

Fig. 10B